

| | | |
|---|--|---------------------------------|
| Name of Student: Pushkar Sane | | |
| Roll Number: 45 | | Lab Assignment Number: 2 |
| Title of Lab Assignment: To implement Python programs with conditionals and loops. | | |
| DOP: 12-09-2023 | | DOS: 28-09-2023 |
| CO Mapped: CO1 | PO Mapped: PO3, PO5, PSO1, PSO2 | Faculty Signature: |

Practical No. 2

Aim: To implement Python programs with conditionals and loops.

Description:

- **Loops**

A loop in programming is a control structure that allows a set of instructions to be executed repeatedly based on a certain condition or for a specified number of times. Loops are essential for automating repetitive tasks, iterating over data structures, and controlling the flow of a program. They help streamline code and improve program efficiency by eliminating the need to write the same code multiple times. In addition to standard loops like for and while loops, Python also provides loop control statements like continue, break, and pass, which offer more fine-grained control over loop execution. These various loop constructs are fundamental for building flexible and efficient Python programs.

1. for Loop:

- The `for` loop is a fundamental looping construct used to iterate over a sequence of items. It is often used when you know how many times you want to repeat a block of code.
- You can use it to loop over sequences like lists, tuples, strings, dictionaries, and other iterable objects.
- The loop assigns each item from the sequence to a variable, and then the code block inside the loop is executed for each item.
- Example:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit) # Output: apple banana cherry
```

2. while loop:

- The `while` loop is used for situations where you don't know in advance how many times the loop should run. It continues to execute as long as a specified condition is `True`.

- b. Be cautious with `while` loops to avoid infinite loops, where the condition is never `False`.

- c. Example:

```
count = 0
while count < 5:
    print(count) #Output: 0 1 2 3 4
    count += 1
```

3. break Statement:

- a. The break statement is used to exit a loop prematurely, even if the loop condition is still True.
- b. It's often used when you want to terminate a loop based on a certain condition.

- c. Example:

```
for i in range (10):
    if i == 5:
        break
    print(i) #Output: 0 1 2 3 4
```

4. Nested Loops:

- a. Python allows you to nest loops within each other, creating multi-level loops. This is useful when you need to iterate through multiple dimensions or combinations of items.

- b. Example:

```
for i in range (3):
    for j in range (2):
        print(f"({i}, {j})") #Output: (0, 0) (0, 1) (1, 0) (1, 1) (2, 0) (2, 1)
```

- **Loop Control Statements**

- 1. continue Statement:**

- a) The `continue` statement is used to skip the current iteration of a loop and move on to the next iteration.
- b) It's useful when you want to skip certain items or perform conditional skipping within a loop.

- c) Example:

```
for i in range(5):  
    if i == 2:  
        continue # Skip the current iteration when i is 2  
    print(i) # Output: 0 1 3 4
```

- 2. else Clause with Loops:**

- a) Python supports the `else` clause with `for` and `while` loops. The code in the `else` block is executed when the loop completes normally, i.e., when the loop condition becomes `False` or there are no more items to iterate.

- b) Example:

```
for i in range(5):  
    print(i) # Output: 0 1 2 3 4  
else:  
    print("Loop finished normally.") # Output: Loop finished normally.
```

- 3. pass Statement:**

- a) The `pass` statement is a no-op, which means it does nothing. It is often used as a placeholder when a statement is syntactically required, but you don't want any code to be executed.

- b) Example:

```
for i in range(3):  
    pass # No output, as the pass statement does nothing
```

These loops and loop control statements provide you with the flexibility to control the flow of your programs and perform repetitive tasks efficiently in Python.

- **Conditional Statements**

Conditional statements in Python allow you to control the flow of your program based on certain conditions. They enable you to make decisions and execute different blocks of code depending on whether a condition is true or false. Here are some of the main conditional statements in Python with examples:

1. if statements:

- a) The `if` statement is used to execute a block of code only if a specified condition is true.
- b) It can be followed by optional `elif` (else if) and `else` clauses for handling multiple conditions.
- c) Example:

```
x = 10
if x > 5:
    print("x is greater than 5") # Output: x is greater than 5
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

2. if Expression (Ternary Operator):

- a) The ternary operator (`x if condition else y`) allows you to write a concise one-liner to assign a value to a variable based on a condition.
- b) Example:

```
age = 25
category = "Adult" if age >= 18 else "Minor"
print(category) # Output: Adult
```

3. while Loop with Condition:

- a) The while loop repeatedly executes a block of code as long as a specified condition is true.
- b) Example:

```
count = 0
```

```
while count < 5:  
    print(count) # Output: 0 1 2 3 4  
    count += 1
```

4. for Loop with Condition:

- a) The for loop can iterate over a sequence or iterable, and you can use the `if` statement within the loop to perform conditional actions.

- b) Example:

```
numbers = [1, 2, 3, 4, 5]  
for num in numbers:  
    if num % 2 == 0:  
        print(f"{num} is even") # Output: 2 is even 4 is even  
    else:  
        print(f"{num} is odd") # Output: 1 is odd 3 is odd 5 is odd
```

5. assert Statement:

- a) The assert statement is used for debugging purposes to test if a given condition is `True`. If the condition is `False`, it raises an `AssertionError` exception.

- b) Example:

```
x = 10  
assert x > 5, "x must be greater than 5" # No output if the assertion is true
```

6. in Operator:

- a) The `in` operator checks if a value exists in a sequence or collection. It is often used in `if` statements for membership testing.

- b) Example:

```
fruits = ["apple", "banana", "cherry"]  
if "banana" in fruits:  
    print("Banana is in the list") # Output: Banana is in the list
```

7. Nested Conditionals:

- a) You can nest conditional statements within each other to handle more complex decision-making scenarios.

b) Example:

```
x = 10
```

```
if x > 5:
```

```
    if x % 2 == 0:
```

```
        print ("x is greater than 5 and even")
```

```
    # Output: x is greater than 5 and even
```

```
else:
```

```
    print ("x is greater than 5 and odd")
```

Conditional statements are a fundamental part of programming, allowing you to create dynamic and responsive code that reacts to different situations. They are crucial for building logic and control in your Python programs

1) To find all the prime numbers in the interval 0 to 100.**Code:**

```
x = 1
```

```
y = 100
```

```
print("Prime numbers between", x, "and", y, "are:")
```

```
for num in range(x, y + 1):
```

```
    if num > 1:
```

```
        for i in range(2, num):
```

```
            if (num % i) == 0:
```

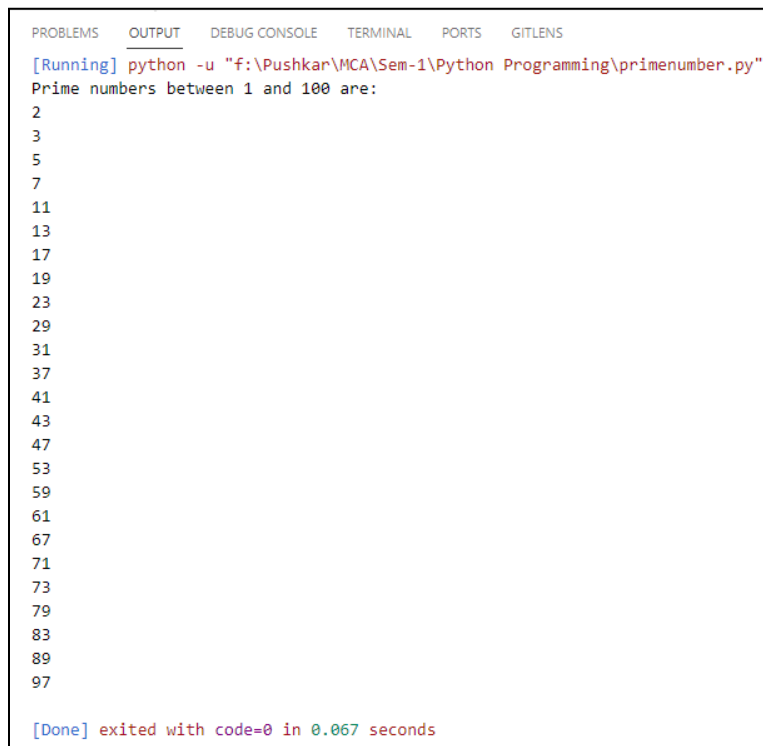
```
                break
```

```
        else:
```

```
            print(num)
```

Conclusion:

Successfully demonstrated finding out prime numbers between 1 and 100

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\primenumber.py"
Prime numbers between 1 and 100 are:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97

[Done] exited with code=0 in 0.067 seconds
```

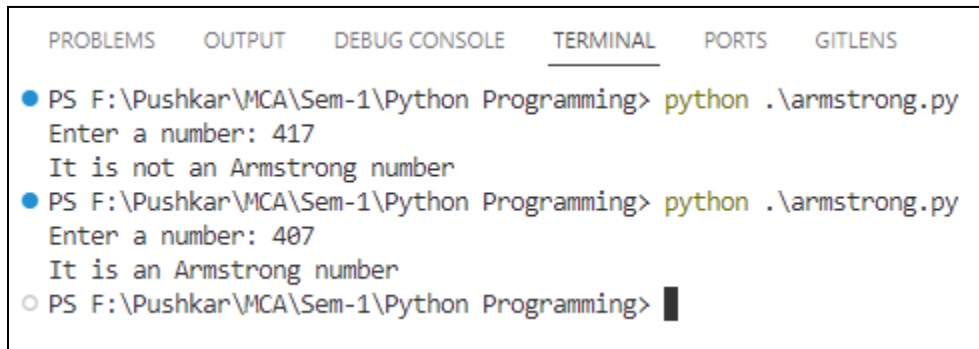

2) To check if the given number is an Armstrong number or not.**Code:**

```
num = int(input("Enter a number: "))
sum = 0
temp = num

while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp //= 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

Conclusion:

Successfully executed the program for finding if the number is an armstrong number or not.

Output:A screenshot of a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and GITLENS. The TERMINAL tab is active. It shows two successful runs of a Python script. In the first run, the user enters 417 and the output is 'It is not an Armstrong number'. In the second run, the user enters 407 and the output is 'It is an Armstrong number'. The prompt is 'PS F:\Pushkar\MCA\Sem-1\Python Programming>' and the command is 'python .\armstrong.py'.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\armstrong.py
Enter a number: 417
It is not an Armstrong number
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\armstrong.py
Enter a number: 407
It is an Armstrong number
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

3) To check if the given char is a vowel or consonant.**Code:**

```
char = input("Enter an alphabet: ")
if char == 'A' or char == 'a' or char == 'I' or char == 'i' or char == 'O' or char == 'o' or
    char == 'U' or char == 'u':
    print(char + " is a vowel")
else:
    print(char + " is a consonant")
```

Conclusion:

Demonstrated the program for checking if the given character is a vowel or a consonant.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

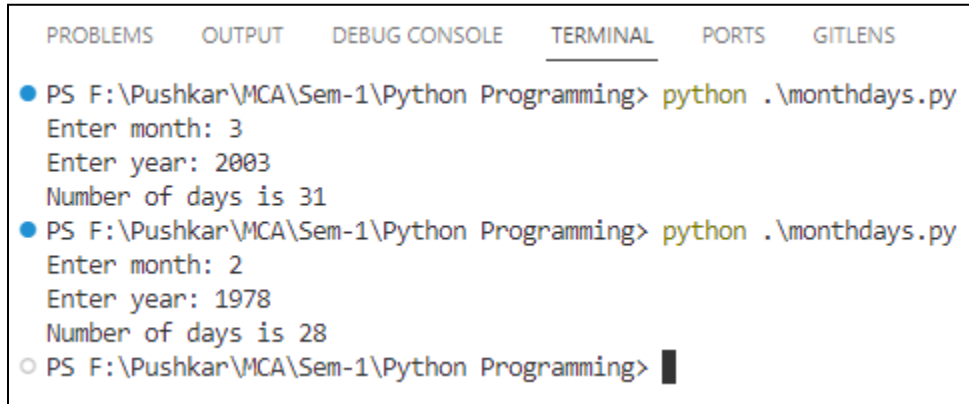
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\vowel.py
Enter an alphabet: A
A is a vowel
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\vowel.py
Enter an alphabet: a
a is a vowel
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\vowel.py
Enter an alphabet: x
x is a consonant
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

4) To convert a month to a number of days.**Code:**

```
import calendar
month = int(input("Enter month: "))
year = int(input("Enter year: "))
num_days = calendar.monthrange(year, month)[1]
print("Number of days is", num_days)
```

Conclusion:

Demonstrated the program for converting the month of a year to the number of days.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

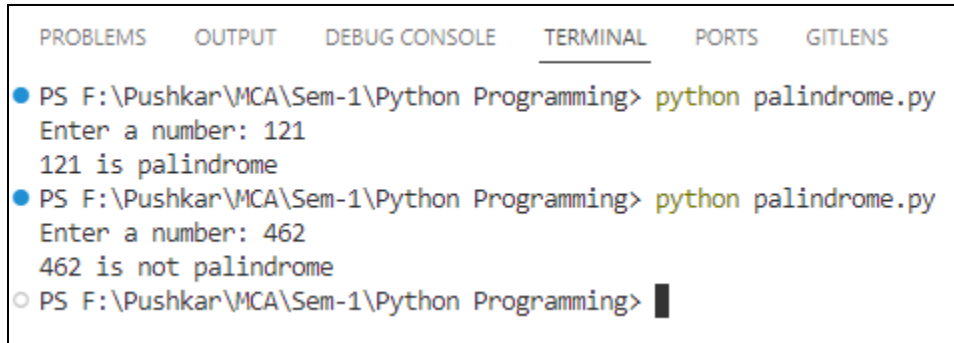
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\monthdays.py
Enter month: 3
Enter year: 2003
Number of days is 31
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\monthdays.py
Enter month: 2
Enter year: 1978
Number of days is 28
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

5) To check if a number is palindrome or not.**Code:**

```
num = int(input("Enter a number: "))
temp = num
reverse = 0
while temp > 0:
    remainder = temp % 10
    reverse = (reverse * 10) + remainder
    temp = temp // 10
if num == reverse:
    print(num, "is palindrome")
else:
    print(num, "is not palindrome")
```

Conclusion:

Demonstrated the program for checking if the given number is palindrome or not.

Output:

The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and GITLENS. The TERMINAL tab is active. It shows three command-line interactions: 1. Running 'python palindrome.py' prompts for 'Enter a number: 121' and outputs '121 is palindrome'. 2. Running 'python palindrome.py' prompts for 'Enter a number: 462' and outputs '462 is not palindrome'. 3. The prompt 'PS F:\Pushkar\MCA\Sem-1\Python Programming>' is shown with a cursor.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
PS F:\Pushkar\MCA\Sem-1\Python Programming> python palindrome.py
Enter a number: 121
121 is palindrome
PS F:\Pushkar\MCA\Sem-1\Python Programming> python palindrome.py
Enter a number: 462
462 is not palindrome
PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

6) Write a program to take in the marks of 3 subjects and display the grade.

Code:

```
sub1 = int(input("Enter marks of English: "))
sub2 = int(input("Enter marks of Maths: "))
sub3 = int(input("Enter marks of Science: "))
```

```
avg = sub1 + sub2 + sub3 / 3
```

```
if avg >= 50:
```

```
    print("You got A grade.")
```

```
elif 40 <= avg < 50:
```

```
    print("You got B grade.")
```

```
elif 30 <= avg < 40:
```

```
    print("You got C grade.")
```

```
elif 20 <= avg < 30:
```

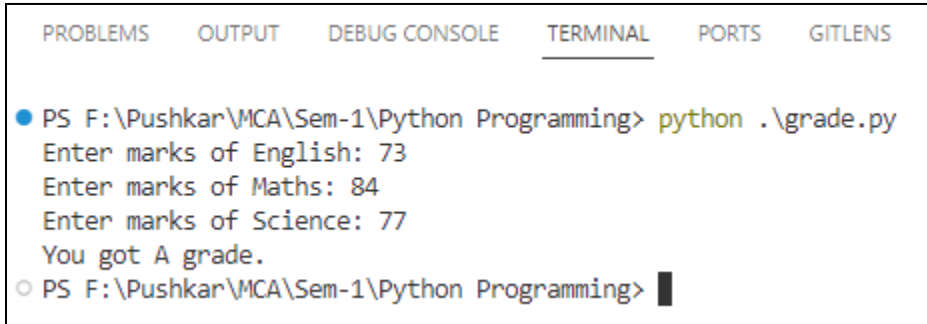
```
    print("You got D grade.")
```

```
else:
```

```
    print("You got failed.")
```

Conclusion:

Demonstrated the program for displaying grade on the basis of marks of 3 subjects.

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\grade.py
Enter marks of English: 73
Enter marks of Maths: 84
Enter marks of Science: 77
You got A grade.
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

7) To add a matrix.**Code:**

```
X = [[12, 7, 3],
      [4, 5, 6],
      [7, 8, 9]]
```

```
Y = [[5, 8, 1],
      [6, 7, 3],
      [4, 5, 9]]
```

```
result = [[0, 0, 0],
           [0, 0, 0],
           [0, 0, 0]]
```

```
for i in range(len(X)):
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]
```

```
for r in result:
    print(r)
```

Conclusion:

Demonstrated the program for adding 2 matrices.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
[Running] python -u "f:\Pushkar\MCA\Sem-1\Python Programming\matrix.py"
Matrix 1:  [[12, 7, 3], [4, 5, 6], [7, 8, 9]]
Matrix 2:  [[5, 8, 1], [6, 7, 3], [4, 5, 9]]

[17, 15, 4]
[10, 12, 9]
[11, 13, 18]

[Done] exited with code=0 in 0.05 seconds
```

- 8) To check the validity of password input by users, give 3 chances to the user to enter the correct password.

Validation:

- a) At least 1 letter between [a-z] and 1 letter between [A-Z].
- b) At least 1 number between [0-9].
- c) At least 1 character from [\$#@].
- d) Minimum length 6 characters.
- e) Maximum length 16 characters.

Code:

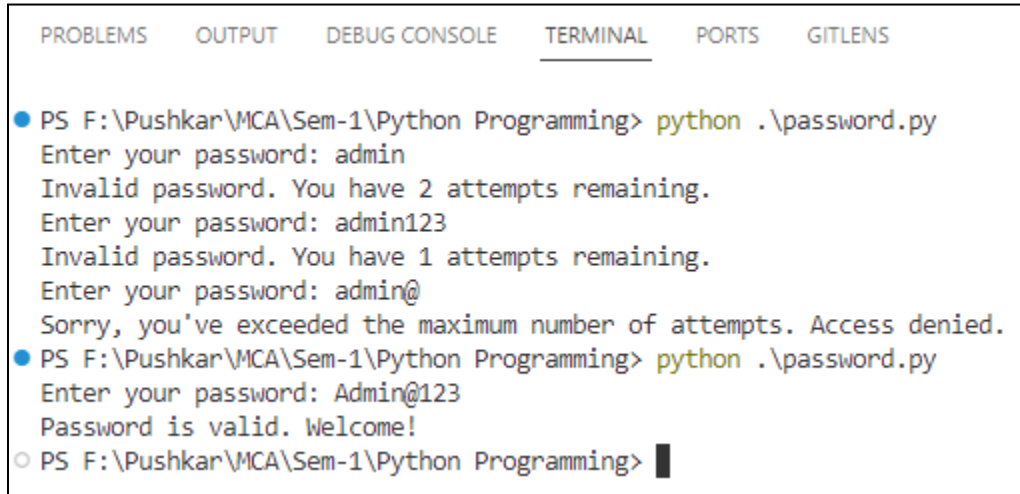
```
import re
max_attempts = 3
while max_attempts > 0:
    password = input("Enter your password: ")
    if 6 <= len(password) <= 16 and re.search("[a-z]", password) and
re.search("[A-Z]", password) and re.search("[0-9]", password) and re.search("[$#@]",
password):
        print("Password is valid. Welcome!")
        break
    else:
        max_attempts -= 1
        if max_attempts > 0:
            print(f"Invalid password. You have", max_attempts, "attempts remaining.")
```

else:

print("Sorry, you've exceeded the maximum number of attempts. Access denied.")

Conclusion:

Demonstrated program of checking the validity of password given by user.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\password.py
Enter your password: admin
Invalid password. You have 2 attempts remaining.
Enter your password: admin123
Invalid password. You have 1 attempts remaining.
Enter your password: admin@
Sorry, you've exceeded the maximum number of attempts. Access denied.
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\password.py
Enter your password: Admin@123
Password is valid. Welcome!
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```